Computer Science 360 Spring 2016
Assignment 1 – Shell
Due January 31th at 11:59pm

Introduction

In this assignment you will implement a simple shell program similar to the Unix shell bash or the command prompt in Windows.

The shell will support execution of programs, the ability to change directories and background execution.

You may implement your solution in C or C++.

It is your responsibility to verify that your submission executes correctly on `linux.csc.uvic.ca`

Neither the TA, nor the instructor, will accept the justification "but it worked on my machine" if you lose marks because your assignment does not run correctly on `linux.csc.uvic.ca`

Be sure to study the man pages for the various functions suggested in this assignment. The functions are in section 2 of the man pages, so you should type (for example):

```
$ man 2 waitpid
```

Requirements

Part I – Basic execution (5 marks)

Using `fork()` and `execvp()` implement the ability for the user to execute arbitrary commands using your shell.

For example, if the user types:

```
shell> ls -l /usr/bin
```

Your shell should execute the **`ls`** command with the parameters **`-l`** and **`/usr/bin`**, which should result in the contents of the directory **`/usr/bin`** being displayed on the screen.

Part II – Changing Directories (5 marks)

Using the functions `getcwd()` and `chdir()` add functionality so that users can change directories using the command `cd`, and print the current working directory using the command `pwd`.

The `cd` command should take one argument that is the directory to change into. The special parameter `..` indicates that the current directory should move "up" one directory. That is, if the current directory is `/home/jason/foo` and the user types `cd ..` the new directory will be `/home/jason`

The `pwd` command takes no parameters.

Additionally, you should change the shell prompt shown in part 1 to include the current working directory:

```
/home/jason/csc360>
```

Part III – Background Execution (5 marks)

NOTE:

It is helpful when testing this portion of your assignment to have a process that does not terminate. **DO NOT** write a program like the following:

```
void main ( void ) {
     while (1)
          ;
}
```

Instead, use something like:

```
#include <unistd.h>

void main ( void ) {
     while (1) {
          sleep (10);
     }
}
```

Even less work is to use the program `/bin/sleep` which takes a single integer parameter which is the number of seconds to sleep for.

Many shells allow programs to be started in the background – that is, the program is run but the shell continues to accept input from the user.

You will implement a simplified version of background processing that supports a fixed number (in this case 5) of processes executing in the background.

If the user types: `bg cat foo.txt` your shell will start the command `cat` with the argument `foo.txt` in the background. That is, the program will execute and the shell will also continue to execute.

The command `bglist` will display a listing of all the commands currently executing in the background, similar to:

```
0: /home/jason/a1/foo
1: /home/jason/a1/foo
Total Background jobs: 2
```

In this case, there are 2 background jobs, both running the program `foo`.

The command `bgkill 1` will send the TERM signal to job 1 to terminate that job. See the man page for the `kill()` system call for details.

Your shell must indicate to the user when background jobs have terminated. Read the man page for the `waitpid()` system call. I suggest using the `WNOHANG` option.

Part IV – Suspending and Resuming background jobs (5 marks)

Many shells allow the user to suspend and resume jobs that are running. You will implement a simplified version of this functionality which allows the user to suspend and resume background jobs only.

The command `stop` will suspend a job that is currently executing in the background by sending it the SIGSTOP signal.

The command `start` will resume a job that is currently stopped by sending it the SIGCONT signal.

Both the `start` and the `stop` command take a single parameter which is the job number to start or stop respectively. You should output an error message if the user tries to stop a job that is currently stopped or start a job which is currently running.

You should update your `bglist` command to include a new field which is the status of the current background processes, where R indicates running and S indicates stopped:

```
0[R]: /home/jason/a1/foo
1[S]: /home/jason/a1/foo
Total Background jobs: 2
```

The GNU Readline Library

We could use the simple **`gets`** to get input from the user, but instead we are going to use the GNU Readline Library. This library allows the user to edit their command line using the standard bash function keys. For example:

- the arrow keys move forwards and backwards
- control-A moves to the front of the line
- control-E moves to the end of the line
- control-K erases the rest of the line
- pressing the TAB key will perform filename completion

The sample code shows how to use the library. The library supports many more options (including command line history) but you are not responsible for those features.

Note: The Readline library allocates memory for the line entered by the user. It is up to the caller (you) to free the memory.

Compilation

You've been provided with a Makefile that builds the sample code. It takes care of linking in the GNU Readline library for you.

Assumptions

To make your code easier to write, you may make the following assumptions:

1. The length of the current working directory is less than 256 characters
2. The maximum number of parameters to any command is less than 15
3. There will be at most 5 background jobs

Submission

Submit a tar archive named `assign1.tar` of your assignment using connex. You must submit a Makefile. If you do not add any additional files, the sample Makefile will suffice.

You can create a tar archive of the current directory by typing:

```
    tar cvf assign1.tar *
```

Please do not submit .o or executable files. Erase them before creating the tar archive.

Note

This assignment is to be done individually. You are encouraged to discuss the design of the solution with your classmates, but each student must implement their own assignment.

The markers will submit your code to an automated plagiarism detection service.

Each student will perform a demonstration of their submitted code. During this demonstration you will be asked to discuss how your code works.

Grading

Your assignment will be graded as follows:

Your program compiles and runs	1 mark
Foreground Execution	5 marks
Background Execution	5 marks
Directories	5 marks
Suspend and Resume	5 marks